PI3 Beowulf Cluster Setup

by: Les Howell

The basic premise of a Beowulf Cluster is to have multiple processors break down a complex calculation into subparts that can be computed exclusively on separate processors, or to compute options of a calculation in parallel to reduce the time waiting for a conclusion. The PI 3 B and B+ make only barely reasonable processors for this use mostly due to the limitations of available RAM memory and networking speed (30MB/s on the B and 300MB/s on the B+), however, algorithms developed may be ported to larger, faster clusters once the software has been developed and debugged. Generally the sectioned algorithms will fit easily within the PI, but the data is the larger issue.

The cluster capability is very limited with limited network speed, because data and algorithms must be generally passed over that network for processing.

In the configuration I am about to introduce, we use MPIch as the processing support for the processor messaging, and a local hardwired network through an unmanaged switch for the cluster. The master may be controlled via the wifi input. The logging and messaging is enabled via secure shell, which requires the installation of ssh, setup of the static IP address in dhcpd, and use ssh-keygen to generate the secure private and public keys to enable automatic login. A common directory is shared by hosting it on the master node and using networked file service to allow the other systems to read it via nfs. Power to the cluster can be provided a few different ways, but to simplify our setup we will utilize a USB power hub which provides roughly 3A/ port.

To allow the most memory, I use Raspian Lite, although other minimal linux installs could be used as well. The memory reserved for video is minimized to again preserve most memory for calculations. The slave systems are set for command line login because they do not need or have a video port.

Program division is aided by the selected messaging service, which can access the node number relative to the total cluster, and that can be read in the software so that the correct apportionment of data can be done.

The commands in this instruction are VERY POWERFUL. Check each command carefully before hitting enter!! dd especially will have the potential to damage your system if not handled carefully. FOR THAT COMMAND verify it carefully before hitting enter. MAKE SURE “of=” points to the sdcard!!

The hardware required is:

- 2 or more Raspberry PI 3 B or B+ systems (5 used here B+ recommended.)
- heatsinks for the Pis :heatsinks
- USB Power hub for the number of PI’s chosen + fans
    - I can’t find the one I bought after three attempts, but you want 3A/port if you can find it. This might work:

      charging hub
- 2 5v ~4” (60mm) fans
    - something like this: 4" 5v fan
- Unmanaged Network Switch for the number of PI’s in the cluster
    - Something like this: network switch
- USB power cables A to micro to fit the hub to pi connectors
    - I cannot find the ones I have. I bought a 5 pack for something like $15. Get extras, we will cut and modify one to power the fans and network switch.
- CAT 5 cables from the switch to each PI
    - Something like this: cat 5 1' cables
- 1 Raspberry PI display and case
    - This is the new and better case: case
    - NOTE: original touch screen case I have, the screen is upside down and affects viewing, so the new case is supposed to fix that.
    - There are others, but some have some issues with connecting, so I used this one: touch screen
- 1 USB mini controller with touch pad and keyboard
    - I got this one: keyboard
- 16G or larger microSD class 10 at least for slaves
    - 16G sdhc1
- 1 64G microSD. Highest class you can get.
    - 64G sdxc These require reformatting to FAT32 for use with the image files provided and the raspberry PI 3B+.
- a bag of 2.5mm nylon standoff’s similar to:
    - 2.5mm standoffs
- A bag of medium nylon wire ties to tidy things up and strap on the fans.

- ** these items are for setup only **
- 1 HDMI cable (for setup and debug)
- 1 LCD display (for setup and debug)
- 1 full sized keyboard and mouse

Software we will be using:

- Raspian on the master (you can use other Linux versions, but the instructions will have to be modified to match)
- RaspianLite on the slaves (you can use others, but the instructions will have to be modified to match)
- ssh-keygen (free)
- MPIch (free downloaded from source forge or current opensource source. Double check that it is virus free.)
- nfs-common (free downloaded via apt-get install nfs-common)
- Fortran (free downloaded via apt-get install gfortran) needed for some libaries.
- Python (free from apt-get) used for demos and checks
- other languages as you want

OK, lets go. First get the hardware together.

Next download the raspian and raspian lite OS's from the Raspberry PI website:Raspian

I will post the raspian lite configured for 8G sd cards on the club website, already basically configured, but you have to change some things as I will note later.

If you use Windows, you will have to find the Raspian instructions for Windows. If you use Linux, as I do, the following will work for you. Don't worry if you normally use Windows, the only difference for you in this entire process is just the initial installation of raspian onto the sd cards. Once that is done, we will use the PI's to set themselves up.

Now lets build the physical part of the cluster.

Install the heat sinks on your PI's. During cluster operations, if you do a good job with the software, the pi will be working hard.

We need the heatsinks and fans to prevent overheating the stack. Clean the large IC and the second largest IC’s with alcohol, either a swab or a cloth with rubbing alcohol will work. Wipe the top with a clean dry lint free cloth like an old tee shirt. Then remove the tape covering and put the heatsink carefully in place. CAREFULLY. Once stuck, leave it, do not try to slide it, or you may break the connections on the bottom of the chip.

Now install one set of standoffs on one board, and see if you get at least 1/8” clearance when the next board is placed on it. This is airflow space for the fans. You may need two standoff’s, I did. Once you know, connect all but one of the boards together with the standoff’s.

Later we will mark and drill the case where we will mount the PI stack to the switch.

Now the remaining board will be the master. Lets set it up first.

From here use the windows instructions from the raspeberry pi site. Or if using linux, follow my instructions.

I am writing this using sudo, but you may use su instead if your operating system requires that.

Fire up your linux system and open a terminal window. We will use terminal windows a lot for these instructions.

First without su or sudo, run this command:

```
$ ls /dev/sd* > temp.txt
```

if you open temp.txt you will see all the drives.

Now insert the 64G sd card and run this command

```
$ ls /dev/sd* > temp1.txt
```

if you open temp1.txt, and compare it to temp.txt, you may detect your sd card entry. But no fear we want the difference, so we now run this command:

```
diff –text temp1.txt temp2.txt
```

and you should see something like:

```
$ diff --text temp.txt temp1.txt

13a14,15

> /dev/sdc1

> /dev/sdc2
```

In this case the sd card I used has two partitions, so they show up as sdc1 and sdc2. Yours probably will show just sdc1.

So /dev/sdc is your sd card.

If the card is formatted with EXTFAT, you cannot use it to copy raspian. Check the format on Widows. On Linux, defaults do not recognize EXTFAT, so opening the file manager will show an error. Format the EXTFAT cards to FAT32 using the following command, substituting the location of your sdcard for dev/sdc, which is where it appeared on my system:

```
$ sudo mkdosfs -F32 <dev/sdc>
```

The .img file is likely zipped from the download something like:

```
bPlusSlave.img.gz
```

Use the gunzip utility to unzip the image.

To install the full Raspian image onto the sdcard, you can use dd as follows:

```
sudo dd if=xxxxxx/raspiandownloadname of=/dev/sdc
```

```
$ sudo dd if=<pathtoRaspian> of=</dev/sdc>
```

where xxx is the path to the download location on your system and raspiandownloadname needs to be the actual name as recorded on your system.

This may take a while, because sd cards are “flash” memory, which means they require some time for the actual write to be made permanent, and the file is 3-8Gigabytes depending on which file you chose to use. Typically on my system, most SD cards actually write and check each block, so through put is about 4.1GB/second, not fast, but you have to realize that each block waits for several hundred microseconds for the full write into the permanent memory. Then is read back for verification of a good write.

Once this is done your separate raspberry pi which will be our system master is ready to boot.

When the write finishes, your cursor will return, so you can open your file browser and look for the card which will show up as probably boot and rootfs. Do not write anything to this card.

Now eject the card using the arrow button next to the Boot or root on the file browser, either one will do the trick, and in a few seconds the entries will disappear from the browser and if you have notifications set, you will get the notification that it is safe to remove the card.

When the rootfs system disappears from the browser wait a second or two for the card to clear, then remove the card, and put it in your pi.

Now connect the pi to the power hub, to the display using the hdmi cable and to the keyboard and mouse. You will want the full size keyboard if possible because typing code on the portable keyboard is not fast or easy.

Turn the power hub on and you should see the pi boot up. If it doesn't boot double check your command sequence. If all that was good refer to troubleshooting the boot issue on the Raspberry PI website. I have had some trouble with B+ systems booting due to excessive power draw. You will see a four color large windows type icon appear which means insufficient power. I have had to either use a different power supply or put the sdcard into a Raspberry PI 3B and boot it and expand then run update before it would boot the Raspberry PI 3B+. But the touch screen and/or the portable keyboard will do the job if you want to install them now. Once it boots and works you should be golden on the B+ so you should be able to plug the card into the B+ and get it to boot.

While the pi is booting we can discuss some more about the setup. The unmanaged network switch means the systems have to have their own static IP set. We need to decide what our numbering system will be. This network will NOT be visible to the outside world. However we want to be consistant to the outside world, but we do not want to overlap the network of your wifi. So, on your linux system or windows system, check out your ip address. Use the following command in a console window:

```
$ ifconfig
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.4  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::d79b:1956:f167:2f6c  prefixlen 64  scopeid 0x20<link>
        ether d8:50:e6:5b:cd:af  txqueuelen 1000  (Ethernet)
        RX packets 1095557  bytes 1340918848 (1.2 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 755878  bytes 97585088 (93.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The approximate equivalent for our windows friends is ipconfig.

I don't have a windows pc up right now, so you will have to check it out.

The 192.168.0 is the network for my local router. The final .4 is the address of this computer on that network. Basically the 192.168 bit is sort of like public, that means it can be used everywhere. .0 means the local subnet, and .4 is my computer.

On our isolated network, lets set up some addresses like this:

```
192.168.1.200 ppiM ppiM.local ppim.lan

192.168.1.201 ppi0 ppi0.local ppi0.lan

192.168.1.202 ppi1 ppi1.local ppi1.lan

192.168.1.203 ppi2 ppi2.local ppi2.lan

192.168.1.204 ppi3 ppi3.local ppi3.lan
```

In this case ppiM is our master, ppi0-3 are slaves .lan tells the system to look on the wired network instead of the wifi network. I already used pim and rpi0-4 on the Pi3B cluster, so I wanted this one to be different and reflect the Plus, so Ppim and Ppi0-4. Ethernet doesn't recognize capitals, so these will appear on the network as ppim and ppi0-4.

OK, if the PI has booted, we can now enter this information into the /etc/hosts file. Log into the pi, the username is pi and the passwork is raspberry. You can keep the username if you want, but I would eventually change both username and password. I won't go into that right now, but think about it. When you do change them, put the same username on your slaves to simplify the interface for mpich.

All the commands are now going to be entered in the terminal window. So click on the terminal window to open it.

To set up the networking, we first have to edit /etc/dhcpd.conf. Use this command:

```
sudo nano /etc/dhcpcd.conf
```

Nano is a nice little command line editor for visual terminals. At the bottom you see the commands you can use. The ^ arrow means control key plus the letter key.

Near the top is the hostname entry:

hostname pi

pi is the name I show here, but you want to change it to your master. If you are using my hosts list, you can set it to piM as:

hostname ppiM

Now Use the search function (control w or ^w) to find the words “static IP”. The section we are looking for is:

```
# Example static IP configuration:
#interface eth0
#static ip_address=192.168.1.172/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8
fd51:42f8:caae:d92e::1
```

Remove the # symbol from the line:

#interface eth0

and from the

#static ip_address=

lines and change the address to the one you selected earlier.

Just above the

# Example static IP configuration:

insert

interface wlan0

This will enable dhcp on the wlan 0 so it can connect to your home or office network. Note that you will have to add any password before you can get this connection.

This completes our changes to dhcpd.conf. Now enter ^x then y and enter to save the changes to the file. Next type the command

```
$ more /etc/dhcpd.conf
```

and verify that the changes were saved.

Here is the example file as modified on my system:

```
    # A sample configuration for dhcpcd.

    # See dhcpcd.conf(5) for details.



    # Allow users of this group to interact with dhcpcd via the
    control socket.

    #controlgroup wheel



    # Inform the DHCP server of our hostname for DDNS.
```

```
hostname piM

# Use the hardware address of the interface for the Client ID.

clientid

# or

# Use the same DUID + IAID as set in DHCPv6 for DHCPv4 ClientID
as per RFC4361.

# Some non-RFC compliant DHCP servers do not reply with this
set.

# In this case, comment out duid and enable clientid above.

#duid



# Persist interface configuration when dhcpcd exits.

persistent



Now g# Rapid commit support.

# Safe to enable by default because it requires the equivalent
option set

# on the server to actually work.

option rapid_commit



# A list of options to request from the DHCP server.

option domain_name_servers, domain_name, domain_search,
host_name

option classless_static_routes

# Most distributions have NTP support.

option ntp_servers
```

```
# Safe to enable by default because it requires the equivalent
option set

# on the server to actually work.

option rapid_commit



# A list of options to request from the DHCP server.

option domain_name_servers, domain_name, domain_search,
host_name

option classless_static_routes

# Most distributions have NTP support.

option ntp_servers

# Respect the network MTU. This is applied to DHCP routes.

option interface_mtu



# A ServerID is required by RFC2131.

require dhcp_server_identifier



# Generate Stable Private IPv6 Addresses instead of hardware
based ones

slaac private



interface wlan0

# Example static IP configuration:

interface eth0

static ip_address=192.168.1.172/24

#static ip6_address=fd51:42f8:caae:d92e::ff/64

#static routers=192.168.0.1
```

```
#static domain_name_servers=192.168.0.1 8.8.8.8
fd51:42f8:caae:d92e::1


# It is possible to fall back to a static IP if DHCP fails:

# define static profile

#profile static_eth0

#static ip_address=192.168.1.23/24

#static routers=192.168.1.1

#static domain_name_servers=192.168.1.1


# fallback to static profile on eth0

#interface eth0

#fallback static_eth0
```

Now go up and select the networking icon on the toolbar and set up your wifi connection. This is so we can do updates and load software. Select your network and enter the password to connect, and wait for the connection.

Got it? If not double check the password and network selection.

In the terminal again type the command:

```
$ sudo raspi-config
```

This will bring up a configuration menu. It is slightly different from the one that comes from the window bar, which is why I want you to use this one. Use the up/down arrows are used to select the function, enter to operate the function and left right to control the select or back or next buttons at the bottom.

Set the location, the time zone and set the keyboard to the US generic 108 key. Verify that the “~” key actually generates the correct ~ character. There are many keyboards to select from and this seems to be the most reliable key to check. If some keys generate bad characters you may have to search to find the correct decoding, refer to the keyboard manufacturer for best results.

Now apply these settings. Next select the rsize option for the OS. This will resize the SD card partition for rootfs to consume the rest of the 64G sd card. This will force you to reboot.

When the system comes back up we should have wifi capability and we should be able to see the whole sd card by using the following command:

```
$df
```

and you should see that about 64G total is visible in the various sections.

Verify that you do in fact have networking by trying to open the web browser and going to google. If that works, good. If not recheck the wifi settings to get the pi on line.

Now we update Raspian. Type the commands:

```
$ sudo apt-get update
```

and then:

```
$ sudo apt-get upgrade
```

This will take a while. As all the software brings in the latest changes for stability and virus protection.

Now to get to each system on the wired network, we have to edit the /etc/hosts file.

With the latest version of Raspian and many other OS’s, you get a long name that is supposed to be predictable. In that case your LAN interface will have something starting with EN and a whole bunch of other stuff. I don’t know how to handle this on all the versions, but on Raspian, again bring up raspi-config and go to the network tab. Click to change predictable names and set the button to off. You must do this for the rest of the networking and file sharing to work!!

```
$ sudo raspi-config
choose->network
choose->turn off predictable names
```

With that out of the way, we now set up the hosts file for our little cluster...

```
$ sudo nano /etc/hosts
```

If you kept the same numbering and host names as I did, then you will want to just copy and paste my hosts file:

```
127.0.0.1 localhost
::1       localhost ip6-localhost ip6-loopback
ff02::1        ip6-allnodes
ff02::2        ip6-allrouters
```

```
127.0.1.1 ppiM

192.168.1.200 ppiM ppiM.local ppiM.lan
192.168.1.201 ppi0 ppi0.local ppi0.lan
192.168.1.202 ppi1 ppi1.local ppi1.lan
192.168.1.203 ppi2 ppi2.local ppi2.lan
192.168.1.204 ppi3 ppi3.local ppi3.lan
```

now save it by the ^x y enter sequence.

And again,

```
$ cat /etc/hosts
```

to verify your changes.

If all is good, we can now load and set up nfs to share the files we need for cluster computing.

I kept mine in my user directory. It simplifies things I think, because if you have yet another user, you have to map everything in all slave systems appropriately and make sure you don’t hit permissions errors.

So to do it my way, make a local directory which I called cloud (because I inadvertantly copied that from some website.)

```
$ mkdir cloud
```

Since this will be shared, we want read write and execute priveleges on it. Software will be shared this way also:

```
$ chmod 766 cloud
```

Now verify that it is setup correctly:

```
$ ls -al cloud
```

so far, so good.

Now we need nfs-common.

```
$ sudo apt-get install ntfs-common
```

and nfs-server:

```
$ sudo apt-get install ntfs-server
```

and while we are at it we need some other things as well for the cluster to work:

```
$ sudo apt-get install mpich
```

If mpich doesn’t install, it is likely not built yet. It is not difficult, just go to git hub and follow the readme to get the tar.gz file and compile it using make. This takes a bit of time, but is essential, because mpich is the message passing software to set up the parallel processes we want to us and also routes any messages back to the master computer, otherwise you won’t get your results.

```
$ sudo apt-get install python
```

```
$ sudo apt-get install gfortran
```

If any of these do not install, you may have to do something like:

```
$ sudo apt-get whatprovides *fortran*
```

If something fails you may need a library or development files and those can be installed as required.

Also mpich is not available yet for the PI. You will have to download it from github and compile it. It is not difficult:

```
$ wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
```

Now we need to configure and install mpich. The instructions are in the README. Follow those, but here are the steps on the version I got, which is 3.2.1 which is the stable version 8/23/2018.

```
$ tar -xvf mpich*.tar.gz
```

```
$ cd mpich-3.2.1
```

```
$ ./configure --prefix=/home/pi/mpich-install 2>&1 | tee c.txt
```

**** this will run for several minutes.

```
$ sudo make install 2>&1 | tee mi.txt
```

**** this will run for up to an hour or so because it is an extensive build.

Now you need to add the mpich software to the path:

```
$ cd
```

```
$ nano .bashrc
```

scroll to the end of the file and add the following:

```
PATH=/home/pi/mpich-install/bin:$PATH
```

Exit nano with the ctrl-x, y, enter sequence and cat .bashrc to make sure that this has taken effect.

We are getting close. Lets continue setting up the ntfs.

First we edit /etc/exports

```
$ sudo nano /etc/exports
```

Most likely this file doesn’t yet exist, so you will get an empty window. You need to put a line in that is something like:

```
<file_location_to_share> *(rw,sync, no_root_squash,no_subtree_check)
```

in my system this is the entry:

```
/home/pi/cloud *(rw,sync,no_root_squash,no_subtree_check)
```

Edit your entry to reflect the location you have chosen to share your files.

This basically ends the setup for your master. These same commands and setups can be done on most linux systems directly if you want to use a linux master. Equivalent steps exist on Windows, but I am not familiar with them. I quit using Windows about 20 years ago except on work supplied systems and I wasn’t permitted to make these kinds of changes on them.

Setting up the slaves

I will make my sdcard code available for the slaves, which will save a lot of effort, so you can simply use dd as you did to create the master disk to copy the slave OS’s make one sdcard for each slave. Mark each card for the slave it will be driving. This helps later in troubleshooting and other things.

You will have to repeat these steps for each slave to create the correct setup.

To make the configuration easier, use only one slave to do the configuration work. Once the cards are all configured, you can insert them into the correct slaves and your good to go.

To configure each slave’s card:

Turn off the cluster.

Connect one of the raspberry PI 3B+ cards. Connect the monitor, keyboard and mouse, connect power to the cluster power hub.

For each slave’s SD card:

Insert the correct SD card for the slave you are going to configure. Make a note of which slave you are configuring so you don’t get confused if you are interrupted.

Power up the cluster. Note that the master will boot also, but you simply cannot see it unless you have the touch screen installed.

After the slave boots using “pi” and “dd”. Then run sudo raspi-config, set the host name, localize (make sure to setup the wifi connection, because it will let you do some of these commands remotely and also update and upgrade the software) and set the keyboard (check it using the # key), ensure it is set to login cli, and select resize the os and reboot the slave.

When it comes back, login and run sudo apt-get update and sudo apt-get upgrade. Grab a cup of coffee.

OK, now we are go for configuration.

First we configure the nfs mount. Use sudo nano /etc/fstab and insert the following line (you can copy and paste from here.)

192.168.1.200:/home/pi/cloud /home/pi/cloud nfs soft,intr,rw,bg, comment=systemd.automount 0 0

Substitute your master IP address for the “192.168.1.172” and your path for the /home/pi/cloud in both locations. Note that the first one is on the master, and the second one is the location on the slave. Notice also that the username “pi” and the directory MUST BE the same on both. At this time, my mounts are not robust. I will update you when/if I discover the correct syntax to ensure robust mounts.

The addition of the “**, comment=systemd.automount”** appears to have fixed the vexing problem.

Save the file using the ^x y enter sequence. Confirm the file by typing the command cat /etc/fstab and verifying the line is there and correct for your systems.

Go to the user directory on this system using cd. Verify that the location is /home/<your_user> as you used in the fstab second entry. Now I used cloud, and it should be on your system. If you changed the name of the remotely mounted file on the master, you must match it here, also in the same login user and directory.

If cloud or your directory is not there type the command:

$ mkdir cloud

but if you used a different directory, replace cloud with that name.

Type the sudo raspi-config command again and make sure that interfacing options SSH is set. Parallel processing messaging is setup to use ssh, thus processes can be started securely and also no password prompt is passed.

I reboot each time after using raspi-config just to ensure the commanded changes are in place.

Now setup the local cluster network. I used 192.168.1.200 as the PIM and 201-204 for the slaves. This is in the file /etc/hosts

$ sudo nano /etc/hosts

Verify that the entry 171.0.1.1 shows the slave hostname. It should have been set by raspi-config.

After that line, add the following:

```
192.168.1.200   ppim  ppim.local ppim.lan
192.168.1.201   ppi1  ppi1.local ppi1.lan
192.168.1.202   ppi2  ppi2.local ppi2.lan
192.168.1.204   ppi3  ppi3.local ppi3lan
192.168.1.205   ppi4  ppi4.local ppi4.lan
```

If you chose different names, you can also copy these lines from your master system which you have already configured. Use ^x y enter to save these changes and then type the command:

```
$ cat /etc/hosts
```

and verify that the changes are inplace.

Now we need to setup the static local ethernet for the cluster on the slave.

```
$ sudo nano /etc/dhcpcd.conf
```

scroll down to this line:

```
hostname
```

change it to:

```
hostname Ppi3
```

make sure that the hostname matches what you set in raspi-config.

Then scroll down to this line

```
# Example static IP configuration:
```

delete the ‘#’ from the next two lines. They should then look like:

```
interface eth0
static ip_address=192.168.0.10/24
```

change the ip_address to the one for your slave:

```
static ip_address=192.168.1.201
```

Look up what you put in the hosts file and make sure the 20x matches your hosts file entry for this slave.

Now above the line:

```
# Example static IP configuration:
```

add the line:

```
interface wlan0
```

The full file as modified (with the appropriate slave name and ip address from the hosts file) should be:

```
# A sample configuration for dhcpcd.
# See dhcpcd.conf(5) for details.

# Allow users of this group to interact with dhcpcd via the control socket.
#controlgroup wheel

# Inform the DHCP server of our hostname for DDNS.
hostname Ppi3

# Use the hardware address of the interface for the Client ID.
clientid
# or
# Use the same DUID + IAID as set in DHCPv6 for DHCPv4 ClientID as per RFC4361.
# Some non-RFC compliant DHCP servers do not reply with this set.
# In this case, comment out duid and enable clientid above.
#duid

# Persist interface configuration when dhcpcd exits.
persistent

# Rapid commit support.
# Safe to enable by default because it requires the equivalent option set
# on the server to actually work.
option rapid_commit

# A list of options to request from the DHCP server.
option domain_name_servers, domain_name, domain_search, host_name
option classless_static_routes
```

```
# Most distributions have NTP support.
option ntp_servers
# Respect the network MTU. This is applied to DHCP routes.
option interface_mtu

# A ServerID is required by RFC2131.
require dhcp_server_identifier

# Generate Stable Private IPv6 Addresses instead of hardware based
ones
slaac private

interface wlan0

# Example static IP configuration:
interface eth0
static ip_address=192.168.1.203/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8
fd51:42f8:caae:d92e::1

# It is possible to fall back to a static IP if DHCP fails:
# define static profile
#profile static_eth0
#static ip_address=192.168.1.23/24
#static routers=192.168.1.1
#static domain_name_servers=192.168.1.1

# fallback to static profile on eth0
#interface eth0
#fallback static_eth0
```

Do the following commands:

```
$ cd
$ mkdir .ssh
$ cd .ssh
$ ssh-keygen
```

Do not enter a filename or passphrase, just hit the enter key three times. You will receive a bit of text output, and and accept the default location. If you now go into the .ssh directory you will have two files, id_rsa, which is the private key and id_rsa.pub which is the public key. Copy the public key to your currenthostname.pub. In my case the command is:

```
$ cp id_rsa.pub ppi0.pub
```

The reason to do this is to make it match what you will need to add later in the authorized keys file on the master and other places. Also the ssh algorithm looks for the “id_rsa” file to encrypt the passphrase for the communications. If this file is not found, ssh will not work and you will be prompted for passwords.

Finally you need to install mpich on each of the slaves. I tried to set this up on the image I provided, and did the install before building the slave image, but for some reason it doesn’t work. So, here are the instructions again for that part. This takes quite a while, 20 minutes or so for the configure, and even longer for the install, so this is a time consuming process.

```
$ wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz
```

Now we need to configure and install mpich. The instructions are in the README. Follow those, but here are the steps on the version I got, which is 3.2.1 which is the stable version 8/23/2018.

```
$ tar -xvf mpich*.tar.gz
$ cd mpich-3.2.1
$ ./configure --prefix=/home/pi/mpich-install 2>&1 | tee c.txt
```

**** this will run for several minutes.

```
$ sudo make install 2>&1 | tee mi.txt
```

**** this will run for up to an hour or so because it is an extensive build.

Now you need to add the mpich software to the path:

```
$ cd
$ nano .bashrc
```

scroll to the end of the file and add the following:

PATH=/home/pi/mpich-install/bin:$PATH

Exit nano with the ctrl-x, y, enter sequence and cat .bashrc to make sure that this has taken effect.

Now shutdown the system. PI’s sometimes mess up the sdcard when power is turned off while the pi is running.

$ sudo shutdown now

Wait until the monitor shows shutdown.

Power down the HUB!!!

I repeat because this is vital!!

Power down the HUB!!!

And remove this sdcard and make sure it is marked for the slave you want it to be. Repeat from here using the next slave name in your cluster.

Start over for the next slave

Once you have all the slaves setup, you need to fix the authorized_keys file on all the systems. Each of the files we copied up as we configured the slaves ssh-keygen stuff, has the public key followed by a user and host name. So we just need to concatenate them into the authorized_keys file. Lets automate this a bit.

Turn off the cluster, insert all the sdcards into the respective slaves, Hook up their power and hook up their internet cables to the internet switch. If your switch requires configuration, do that now following it’s instructions.

Then move the keyboard and monitor to the master and power up the cluster. It is now almost a cluster. To complete it log into the master, and cd to the .ssh directory.

Use scp (secure copy) copy the public key from each system to the master:

$ scp . pi@ppi#:/home/pi/.shh/*.pub

The # is the slave number.

When you have all of them copied:

Type the following command:

$ cat *.pub > authorized_keys

And now you have your authorized_keys file. Copy this file to all the slaves.

$ scp authorized_keys pi@rpi#:/home/pi/.ssh

And now it is a cluster, and to drive it is the next step.

Test that it is setup correctly by using the following command to ssh to each slave:

```
$ ssh ppi#
```

Where # is replaced by the slave number. You should immediately see the login prompt for the slave number you called and you may see the login message if it is still setup (I did not tell you to remove it, but you can if you wish by following the steps shown in the raspberry PI website.)

+++

This was true on mpich-3.1:

There appears to be a glitch in the implementation of mpich2 near line 1531 with the definition of a structure MPID_Request.

The line is:

```
} MPID_Request ATTRIBUTE((__aligned__(32)));
```

Corrected should be:

```
} ATTRIBUTE((__aligned__(32))) MPID_Request;
```

As documented here:

https://lists.mpich.org/pipermail/discuss/2016-May/004764.html

++

Before we can do any python programming we need the utility mpi4py.

To make it simple to make sure our slaves have a copy, we will do this in the cloud directory. Type the following command:

```
$ cd cloud
$ wget https://bitbucket.org/mpi4py/mpi4py/downloads/mpi4py-2.0.0.tar.gz
$ sudo tar -zxf mpi4py-2.0.0.tar.gz
```

Then we need to setup mpi4py on the master and all slaves. ON each one do the following:

skip this command for the master

```
$ ssh ppi#
```

Then on each one do:

```
     $ cd mpi4py-2.0.0
     $ sudo apt-get python-dev
     $ sudo python setup.py build
     $ sudo python setup.py install
     $ cd
# note the “.”
     $ sudo nano .bashrc
```

Go to the end of the .bashrc file and you should see:

```
     PATH=/home/pi/mpich-install/bin:$PATH
```

add the following line:

```
     LD_LIBRARY_PATH=/home/pi/mpich-install/lib:$LD_LIBRARY_PATH
```

exit nano with ^x,y, enter key sequence and now we are ready!!

Programming the cluster in python

Our first program is a simple Helloworld application written in python, which I enhanced a bit to show the host and thread being used, with random sleep statements on each. The code is :

```
#!/usr/bin/env python
"""
Parallel Hello World
"""
from random import *
import time
from mpi4py import MPI
import sys

t=(random()*3)
time.sleep(t)
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()
```

```
sys.stdout.write(
    "Hello, World! I am process %d of %d on %s slept %f\n"
    % (rank, size, name,t))
```

That’s it. To run this code you use the command:

```
mpiexec -n 4 python helloworld.py
```

That runs it on the 4 processors in the local PI. But to get more capability, you use a hosts file containing the names of the hosts followed by a colon and the number of usable processors in the host. I have left 2 processes available on the master for generic use by the OS and any remote logins.

```
$ cd ~/cloud
$ nano hosts
ppiM:2
ppi1:4
ppi2:4
ppi3:4
ppi4:4
```

Now when you invoke the program like this:

```
mpiexec -n 18 -f hosts python helloworld.py
```

You receive the following output:

```
Hello, World! I am process 5 of 18 on rpi0 slept 0.000536
Hello, World! I am process 6 of 18 on rpi1 slept 0.115505
Hello, World! I am process 10 of 18 on rpi2 slept 0.441105
Hello, World! I am process 17 of 18 on rpi3 slept 0.685320
Hello, World! I am process 4 of 18 on rpi0 slept 0.834156
Hello, World! I am process 0 of 18 on piM slept 1.087638
Hello, World! I am process 7 of 18 on rpi1 slept 1.108027
Hello, World! I am process 3 of 18 on rpi0 slept 1.150823
Hello, World! I am process 8 of 18 on rpi1 slept 1.248916
Hello, World! I am process 9 of 18 on rpi1 slept 1.420054
```

```
Hello, World! I am process 14 of 18 on rpi3 slept 1.525538
Hello, World! I am process 13 of 18 on rpi2 slept 1.649570
Hello, World! I am process 12 of 18 on rpi2 slept 1.720092
Hello, World! I am process 16 of 18 on rpi3 slept 2.163041
Hello, World! I am process 11 of 18 on rpi2 slept 2.216600
Hello, World! I am process 2 of 18 on rpi0 slept 2.287739
Hello, World! I am process 15 of 18 on rpi3 slept 2.320290
Hello, World! I am process 1 of 18 on piM slept 2.884885
```

Since each process sleeps a random time, their execution will vary from run to run. If you put the Raspian command time in front of the mpiexec, you will receive the execution time for the whole process and you will find that the whole thing took only 3.4 seconds. Total sleep time was well over that because it was divided among the 18 processors.

```
$ time mpiexec -n 18 -f hosts python helloworld.py
```

If you get an error, it is likely that the nfs share did not mount on the slave processors. This can be fixed by issuing the following command

```
mpiexec -n 4 rpi0,rpi1,rpi2,rpi3 sudo mount -a
```

which will force the slaves to mount the nfs server. If this does not clear the error, check your command, check that you can successfully ssh to each of the servers from the master, and check that each is capable of the nfs mount. Double check the hosts file on each slave against the master to make sure that the hosts names resolve to the correct ethernet address and that local host is defined as default:

```
127.0.0.1 localhost
```

These are the most common errors. However if it is something else please check the installation process you followed.

Should you find you cannot use the image file I provided for the slaves, you can just follow the procedure for the master on one slave SD card, then either create an image or directly copy the whole SDCARD to another SDCARD using dd. If you do not know how to do this, check the internet for instructions. BE CAREFUL with DD. It will write to whatever disk you tell it, so the “of=” argument is VITAL.